UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/720,726	11/24/2003	Adam G. Wolff	LZLO-01006US0	2756

28554 7590 06/27/2007
VIERRA MAGEN MARCUS & DENIRO LLP
575 MARKET STREET SUITE 2500
SAN FRANCISCO, CA 94105

EXAMINER	
CHEN, QING	

ART UNIT	PAPER NUMBER
2191	

MAIL DATE	DELIVERY MODE
06/27/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/720,726	WOLFF ET AL.	
	Examiner	Art Unit	
	Qing Chen	2191	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *24 November 2003*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-65* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-65* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *16 April 2004* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☒ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This is the initial Office action based on the application filed on November 24, 2003.

2. **Claims 1-65** are pending.

Oath/Declaration

3. The oath or declaration is defective. A new oath or declaration in compliance with 37 CFR 1.67(a) identifying this application by application number and filing date is required. See MPEP §§ 602.01 and 602.02.

The oath or declaration is defective because:
It does not identify the city and either state or foreign country of residence of the fourth inventor. The residence information may be provided on either an application data sheet or supplemental oath or declaration.

Drawings

4. The drawings are objected to as failing to comply with 37 CFR 1.84(p)(5) because they include the following reference character(s) not mentioned in the description:

- Reference number 10 in Figure 1;
- Reference number 316 in Figure 3; and
- Reference number 422 in Figure 4.

Corrected drawing sheets in compliance with 37 CFR 1.121(d), or amendment to the specification to add the reference character(s) in the description in compliance with 37 CFR 1.121(b) are required in reply to the Office action to avoid abandonment of the application.

The drawings are objected to because:

- "Source" is misspelled in Figure 1, Element 10;
- "Optimizer" is misspelled in Figure 1, Element 16;
- "Environment" is misspelled in Figure 1, Element 14;
- "Optimizing" is misspelled in Figure 3;
- "serialized" and "representation" are misspelled in Figure 3, Element 314;
- "01" should be changed to "O1" in Figure 3, Element 302;
- "02" and "03" should be changed to "O2" and "O3," respectively, in Figure 3, Elements 314 and 316;
- "that has *and* ID" should read -- that has *an* ID -- in Figure 3, Element 316; and
- "load runtime in runtime" should presumably read -- execute in runtime -- in Figure 3, Element 320.

Corrected drawing sheets in compliance with 37 CFR 1.121(d) are required in reply to the Office action to avoid abandonment of the application.

Any amended replacement drawing sheet should include all of the figures appearing on the immediate prior version of the sheet, even if only one figure is being amended. The figure or figure number of an amended drawing should not be labeled as "amended." If a drawing figure is to be canceled, the appropriate figure must be removed from the replacement sheet, and where necessary, the remaining figures must be renumbered and appropriate changes made to the brief description of the several views of the drawings for consistency. Additional replacement sheets may be necessary to show the renumbering of the remaining figures. Each drawing sheet submitted after the filing date of an application must be labeled in the top margin as either

"Replacement Sheet" or "New Sheet" pursuant to 37 CFR 1.121(d). If the changes are not accepted by the Examiner, the Applicant will be notified and informed of any required corrective action in the next Office action. The objection to the drawings will not be held in abeyance.

Specification

5. The abstract of the disclosure is objected to because "an source language" should read -- a source language --. Correction is required. See MPEP § 608.01(b).

6. The attempt to incorporate subject matter into this application by reference to copending application no. 10/092,010 is ineffective because the root words "incorporate" and/or "reference" have been omitted. See 37 CFR 1.57(b)(1).

The incorporation by reference will not be effective until correction is made to comply with 37 CFR 1.57(b), (c), or (d). If the incorporated material is relied upon to meet any outstanding objection, rejection, or other requirement imposed by the Office, the correction must be made within any time period set by the Office for responding to the objection, rejection, or other requirement for the incorporation to be effective. Compliance will not be held in abeyance with respect to responding to the objection, rejection, or other requirement for the incorporation to be effective. In no case may the correction be made later than the close of prosecution as defined in 37 CFR 1.114(b), or abandonment of the application, whichever occurs earlier.

Any correction inserting material by amendment that was previously incorporated by reference must be accompanied by a statement that the material being inserted is the material incorporated by reference and the amendment contains no new matter. See 37 CFR 1.57(f).

7. The disclosure is objected to because of the following informalities:

- "Figure 7A" should read -- Figure 6A -- on page 7, paragraph [0030].
- "webserver" should read -- web server -- on page 12, paragraph [0042].
- "render" should read -- renderer -- on page 11, paragraph [0042] and page 21, paragraph [0068].
- "Steps 302, 304, 306, 308 and 310 *general* represent" should read -- Steps 302, 304, 306, 308 and 310 *generally* represent -- on page 12, paragraph [0044].
- "co-pending application serial *no* 10/092,010" should read -- co-pending application serial *no.* 10/092,010 --. Note the added period (.).

Appropriate correction is required.

8. The use of trademarks, such as JAVA, WINDOWS, J2EE, JDBC, FLASH, and JAVASCRIPT, has been noted in this application. Trademarks should be capitalized wherever they appear (capitalize each letter OR accompany each trademark with an appropriate designation symbol, *e.g.*, ™ or ®) and be accompanied by the generic terminology (use trademarks as adjectives modifying a descriptive noun, *e.g.*, "the JAVA programming language").

Although the use of trademarks is permissible in patent applications, the proprietary nature of the marks should be respected and every effort made to prevent their use in any manner, which might adversely affect their validity as trademarks.

Claim Objections

9. **Claims 1-5, 8, 9, 12-16, 21, 23, 24, 32-37, 43, 45, 46, 52, and 63** are objected to because of the following informalities:

- **Claims 1, 3, 12, and 13** recite the limitation "the application." Applicant is advised to change this limitation to read "the computer application" for the purpose of providing it with proper explicit antecedent basis.
- **Claims 2, 4, and 5** depend on Claim 1 and, therefore, suffer the same deficiency as Claim 1.
- **Claims 14 and 15** depend on Claim 13 and, therefore, suffer the same deficiency as Claim 13.
- **Claims 3, 52, and 63** contain a typographical error: a period (.) should be added after the limitation body.
- **Claim 8** recites the limitation "the representation." Applicant is advised to change this limitation to read "the serialized representation" for the purpose of providing it with proper explicit antecedent basis.
- **Claims 9 and 46** contain a typographical error: "extensible markup language" should read -- Extensible Markup Language --.
- **Claim 16** contains a typographical error: "application source" should presumably read -- application source code --.
- **Claim 20** contains a typographical error: "an source language" should read -- a source language --.

- **Claim 23** recites the limitation "the file." Applicant is advised to change this limitation to read "the text file" for the purpose of providing it with proper explicit antecedent basis.
- **Claim 24** recites the limitation "the file." Applicant is advised to change this limitation to read "the Extensible Markup Language file" for the purpose of providing it with proper explicit antecedent basis.
- **Claim 32** contains a typographical error: "created" is misspelled.
- **Claims 33 and 37** recite the limitation "the source server." Applicant is advised to change this limitation to read "the application source server" for the purpose of providing it with proper explicit antecedent basis.
- **Claims 34-36** depend on Claim 33 and, therefore, suffer the same deficiency as Claim 33.
- **Claims 36 and 45** contain a typographical error: "includes the *step* of" should read --includes the *steps* of --.
- **Claim 43** contains a typographical error: "Extensible markup language" should read -- Extensible Markup Language --.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

10. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

11. **Claims 5, 9, 10, 13-15, 17, 18, 21-24, 26, 29-32, 35-51, and 65** are rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claims 5, 26, 35, and 65 contain the trademark or trade name FLASH. When a trademark or trade name is used in a claim as a limitation to identify or describe a particular material or product, the claim does not comply with the requirements of the 35 U.S.C. 112, second paragraph. *Ex parte Simpson*, 218 USPQ 1020 (Bd. App. 1982). The claim scope is uncertain since the trademark or trade name cannot be used properly to identify any particular material or product. A trademark or trade name is used to identify a source of goods, and not the goods themselves. Thus, the use of a trademark or trade name in a claim to identify or describe a material or product (in the present case, a specific media presentation software) would not only render a claim indefinite, but would also constitute an improper use of the trademark or trade name.

Claims 9 and 46 recite the limitation "the serialized application." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the

Examiner subsequently interprets this limitation as reading "the serialized representation" for the purpose of further examination.

Claim 10 depends on Claim 9 and, therefore, suffers the same deficiency as Claim 9.

Claims 10 and 43 recite the limitation "the markup language file." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "a markup language file" for the purpose of further examination.

Claims 13, 29, 37, and 49 recite the limitation "the step of serializing." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the step of creating" for the purpose of further examination.

Claims 14 and 15 depend on Claim 13 and, therefore, suffer the same deficiency as Claim 13.

Claims 30 and 31 depend on Claim 29 and, therefore, suffer the same deficiency as Claim 29.

Claims 50 and 51 depend on Claim 49 and, therefore, suffer the same deficiency as Claim 49.

Claim 17 recites the limitation "the serialized description." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the serialized representation" for the purpose of further examination.

Claim 18 recites the limitation "the runtime." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "runtime" for the purpose of further examination.

Claim 21 recites the limitation "the running application." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the application" for the purpose of further examination.

Claims 22-24 depend on Claim 21 and, therefore, suffer the same deficiency as Claim 21.

Claim 26 recites the limitation "the virtual machine." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "a virtual machine" for the purpose of further examination.

Claim 32 recites the limitation "the serialized file." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the serialized representation" for the purpose of further examination.

Claim 36 recites the limitation "the runtime." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the runtime memory state" for the purpose of further examination.

Claims 38, 40, 48, and 49 recite the limitation "the application." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "an application" for the purpose of further examination.

Claims 39 and 41-47 depend on Claim 38 and, therefore, suffer the same deficiency as Claim 38.

Claims 50 and 51 depend on Claim 49 and, therefore, suffer the same deficiency as Claim 49.

Claims 42 and 43 recite the limitation "the description." There is insufficient antecedent basis for this limitation in the claims. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "a description" for the purpose of further examination.

Claim 45 recites the limitation "the serialization step." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the creating step" for the purpose of further examination.

Claim Rejections - 35 USC § 102

12. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

13. **Claims 1, 2, 4, 6-8, 11-23, 25, 27-33, 36-39, 41, 42, 44, 45, and 47-62** are rejected under 35 U.S.C. 102(e) as being anticipated by **<u>Abbott et al.</u> (US 7,191,441)**.

As per **Claim 1**, <u>Abbott et al.</u> disclose:

- creating a serialized representation of application objects in a runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9:*

10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.");

- building an object code file using the serialized representation *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.")*; and
- providing the computer application to a new runtime environment *(see Column 18: 61-65, "... the suspended application is now ready to run, and control is passed to execution engine of the Java VM (step 860).")*.

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Abbott et al. further disclose:

- reading from a runtime memory space a description of each object of a running application *(see Column 12: 6-10, "Thus the first item is to create a vector or table 510 having three columns. Each object in turn in the heap is then identified from the system Reference queue, and added into the table 510, with one object entry per row of the table.")*.

As per **Claim 4**, the rejection of **Claim 2** is incorporated; and Abbott et al. further disclose:

- wherein the runtime environment is a virtual machine *(see Figure 2: 40)*.

As per **Claim 6**, the rejection of **Claim 1** is incorporated; and Abbott et al. further disclose:

- identifying each object of a running application by a unique identifier *(see Column 12: 10-15, "The first column 511 is used to hold the identifier or reference number of an object, and the third column 513 is used to hold the length of the object. The actual contents of the object can now be copied over into the save file for the snapshot (not shown in FIG. 5).").*

As per **Claim 7**, the rejection of **Claim 6** is incorporated; and Abbott et al. further disclose:

- detaching each object from an object hierarchy and creating a description of each slot in said object *(see Column 12: 29-36, "Therefore, as shown in the sequence of diagrams FIG. 5A, 5B, and 5C, the heap is scanned to detect all inter-object references. In the example of FIG. 5A, two such references are present, one 532A from object A to object S, and one 533A from object F to object S. As each object reference is found, a linked list is built up for that object, which is headed from the second column 512 of the vector table 510.").*

As per **Claim 8**, the rejection of **Claim 6** is incorporated; and Abbott et al. further disclose:

- providing the serialized representation directly to the building step *(see Column 18: 10-14, "... the saved state is loaded by a special "javaload" tool, which loads the stored* *application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM. The reloading of the application is performed in stages.").*

As per **Claim 11**, the rejection of **Claim 1** is incorporated; and Abbott et al. further disclose:

- assigning a serialization identifier to each object *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

As per **Claim 12**, the rejection of **Claim 1** is incorporated; and Abbott et al. further disclose:

- developing the computer application in an interpreted language *(see Column 7: 28-33, "Another component of the Java VM is the interpreter 156, which is responsible for reading in Java byte code from loaded classes, and converting this into machine instructions for the relevant platform.").*

As per **Claim 13**, the rejection of **Claim 6** is incorporated; and Abbott et al. further disclose:

- assigning a function ID to each function in the computer application *(see Column 16: 9-15, "... a table is created containing: the name of the DLL; its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).").*

As per **Claim 14**, the rejection of **Claim 13** is incorporated; and Abbott et al. further disclose:

- creating a function ID table associating each function ID with function code *(see Column 16: 9-15, "... a table is created containing: the name of the DLL; its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*.

As per **Claim 15**, the rejection of **Claim 13** is incorporated; and Abbott et al. further disclose:

- assigning unique function identifiers to functions within closures *(see Column 16: 2-7, "In order to save the state associated with such DLLs, all loaded DLLs are recorded with their base addresses, and if they contain JNI code, all references to them from class loaders and NativeLibrary classes are also recorded (NativeLibrary classes provide the mechanism for a Java application to access the DLLs).")*.

As per **Claim 16**, the rejection of **Claim 1** is incorporated; and Abbott et al. further disclose:

- using a compiled version of application source code compiled prior to the creating step in combination with the serialized representation to build the new object code file *(see Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage* *collection, and forcing compilation of at least some of the application."; Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.").*

As per **Claim 17**, the rejection of **Claim 1** is incorporated; and Abbott et al. further disclose:

- writing the serialized representation to a text file prior to said step of building *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

As per **Claim 18**, the rejection of **Claim 1** is incorporated; and Abbott et al. further disclose:

- wherein the step of creating is performed in runtime *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420).").*

As per **Claim 19**, the rejection of **Claim 1** is incorporated; and Abbott et al. further disclose:

- wherein the step of creating is performed in a different runtime *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved* *(step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420).").*

As per **Claim 20**, Abbott et al. disclose:

- compiling an application provided in a source language *(see Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application.")*;

- initializing the application in a runtime environment *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420).")*; and

- creating a serialized representation of the application *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.").*

As per **Claim 21**, the rejection of **Claim 20** is incorporated; and Abbott et al. further disclose:

- reading from the runtime environment a description of each object of the application *(see Column 12: 6-10, "Thus the first item is to create a vector or table 510 having three columns. Each object in turn in the heap is then identified from the system Reference queue, and added into the table 510, with one object entry per row of the table.").*

As per **Claim 22**, the rejection of **Claim 21** is incorporated; and Abbott et al. further disclose:

- outputting the description to a rebuilder *(see Column 18: 10-14, "... the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM. The reloading of the application is performed in stages.").*

As per **Claim 23**, the rejection of **Claim 22** is incorporated; and Abbott et al. further disclose:

- storing the serialized representation in a text file and providing the text file to the rebuilder *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 18: 10-14, "... the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM. The reloading of the application is performed in stages.").*

As per **Claim 25**, the rejection of **Claim 20** is incorporated; and Abbott et al. further disclose:

- wherein the runtime environment is a virtual machine *(see Figure 2: 40).*

As per **Claim 27**, the rejection of **Claim 20** is incorporated; and Abbott et al. further disclose:

- assigning a serialization identifier to each initialized object *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

As per **Claim 28**, the rejection of **Claim 20** is incorporated; and Abbott et al. further disclose:

- enumerating each object in a global scope and writing a serialized description of each said object *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 12: 6-10, "Thus the first item is to create a vector or table 510 having three columns. Each object in turn in the heap is then identified from the system Reference queue, and added into the table 510, with one object entry per row of the table.").*

As per **Claim 29**, the rejection of **Claim 20** is incorporated; and Abbott et al. further disclose:

- assigning a function ID to each function in the application *(see Column 16: 9-15, "... a table is created containing: the name of the DLL; its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*.

As per **Claim 30**, the rejection of **Claim 29** is incorporated; and Abbott et al. further disclose:

- creating a function ID table associating each function ID with a function call *(see Column 16: 9-15, "... a table is created containing: the name of the DLL; its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*.

As per **Claim 31**, the rejection of **Claim 29** is incorporated; and Abbott et al. further disclose:

- assigning function identifiers to functions within closures *(see Column 16: 2-7, "In order to save the state associated with such DLLs, all loaded DLLs are recorded with their base addresses, and if they contain JNI code, all references to them from class loaders and NativeLibrary classes are also recorded (NativeLibrary classes provide the mechanism for a Java application to access the DLLs).")*.

As per **Claim 32**, the rejection of **Claim 20** is incorporated; and Abbott et al. further disclose:

- combining the serialized representation with an object code file created by said compiling step to create a new object code file *(see Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application."; Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.")*.

As per **Claim 33**, <u>Abbott et al.</u> disclose:

- requesting an application from an application source server *(see Column 5: 63-67, "The middleware also includes Java programming which acts to cause transactions as Java applications 50 to run on top of the Java VM 40. In a typical server environment, multiple Java VMs may be running on computer system 10, in one or more middleware environments.")*; and
- receiving object code of a serialized description of the application from the application source server, the object code including instructions for creating a runtime memory state of the application *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an*

initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.").

As per **Claim 36**, the rejection of **Claim 33** is incorporated; and Abbott et al. further disclose:

- loading the object code into runtime memory state *(see Column 18: 10-14, "... the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM. The reloading of the application is performed in stages.")*; and

- executing the object code *(see Column 18: 14-16, "The various components of the Java VM are provided with restore commands to restart themselves using the saved information.").*

As per **Claim 37**, the rejection of **Claim 33** is incorporated; and Abbott et al. further disclose:

- calling a function using a function identifier from the object code *(see Column 16: 9-15, "... a table is created containing: the name of the DLL; its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).")*; and

- receiving function code from the application source server *(see Column 15: 66-67 through Column 16: 1-2, "Dynamic link libraries (DLLs) are utilised in a Java environment to provide platform functionality for the Java VM itself, and also optionally by applications that contain JNI code.").*

As per **Claim 38**, Abbott et al. disclose:

- creating a serialized representation of application objects in a runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.")*;

- building an object code file using the serialized representation *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.")*; and

- providing an application to a new runtime environment *(see Column 18: 61-65, "... the suspended application is now ready to run, and control is passed to execution engine of the Java VM (step 860).")*.

As per **Claim 39**, the rejection of **Claim 38** is incorporated; and Abbott et al. further disclose:

- reading from a runtime environment memory space a description of each object of a running application *(see Column 12: 6-10, "Thus the first item is to create a vector or table 510* *having three columns. Each object in turn in the heap is then identified from the system Reference queue, and added into the table 510, with one object entry per row of the table.").*

As per **Claim 41**, the rejection of **Claim 38** is incorporated; and Abbott et al. further disclose:

- identifying each object of a running application by a unique identifier *(see Column 12: 10-15, "The first column 511 is used to hold the identifier or reference number of an object, and the third column 513 is used to hold the length of the object. The actual contents of the object can now be copied over into the save file for the snapshot (not shown in FIG. 5).").*

As per **Claim 42**, the rejection of **Claim 41** is incorporated; and Abbott et al. further disclose:

- writing a description to a text file and compiling the text file *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application.").*

As per **Claim 44**, the rejection of **Claim 41** is incorporated; and Abbott et al. further disclose:

- detaching each object from an object hierarchy and creating a description of each slot in said object *(see Column 12: 29-36, "Therefore, as shown in the sequence of diagrams FIG. 5A, 5B, and 5C, the heap is scanned to detect all inter-object references. In the example of FIG. 5A, two such references are present, one 532A from object A to object S, and one 533A from object F to object S. As each object reference is found, a linked list is built up for that object, which is headed from the second column 512 of the vector table 510.").*

As per **Claim 45**, the rejection of **Claim 41** is incorporated; and Abbott et al. further disclose:

- determining whether the object is a class *(see Column 11: 5-8, "... all loaded classes are held as objects 145 on the heap 140, and may contain absolute references to instance objects, which need to be encoded for storage.")*; and
- writing a serialized description of the class *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

As per **Claim 47**, the rejection of **Claim 39** is incorporated; and Abbott et al. further disclose:

- assigning a serialization identifier to each object *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

As per **Claim 48**, the rejection of **Claim 39** is incorporated; and Abbott et al. further disclose:

- developing an application in an interpreted language *(see Column 7: 28-33, "Another component of the Java VM is the interpreter 156, which is responsible for reading in Java byte code from loaded classes, and converting this into machine instructions for the relevant platform.").*

As per **Claim 49**, the rejection of **Claim 41** is incorporated; and Abbott et al. further disclose:

- assigning a function ID to each function in an application *(see Column 16: 9-15, "... a table is created containing: the name of the DLL; its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).").*

As per **Claim 50**, the rejection of **Claim 49** is incorporated; and Abbott et al. further disclose:

- creating a function ID table associating each function ID with function code *(see Column 16: 9-15, "... a table is created containing: the name of the DLL; its load location (memory address); and the NativeLibrary class association (in the case of DLLs containing JNI code).").*

As per **Claim 51**, the rejection of **Claim 49** is incorporated; and Abbott et al. further disclose:

- assigning function identifiers to functions within closures *(see Column 16: 2-7, "In order to save the state associated with such DLLs, all loaded DLLs are recorded with their base addresses, and if they contain JNI code, all references to them from class loaders and NativeLibrary classes are also recorded (NativeLibrary classes provide the mechanism for a Java application to access the DLLs).")*.

As per **Claim 52**, Abbott et al. disclose:

- compiling first object code for an application *(see Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application.")*;

- loading the application into a first runtime environment *(see Column 5: 63-67, "The middleware also includes Java programming which acts to cause transactions as Java applications 50 to run on top of the Java VM 40.")*;

- creating a serialized representation of a memory space in said first runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.")*;

- building second object code using said serialized representation *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code. ")*; and

- deploying said second object code *(see Column 18: 61-65, "... the suspended application is now ready to run, and control is passed to execution engine of the Java VM (step 860). ").*

As per **Claim 53**, the rejection of **Claim 52** is incorporated; and <u>Abbott et al.</u> further disclose:

- wherein the step of compiling is performed on an interpreted language application *(see Column 7: 28-33, "Another component of the Java VM is the interpreter 156, which is responsible for reading in Java byte code from loaded classes, and converting this into machine instructions for the relevant platform. ").*

As per **Claim 54**, the rejection of **Claim 52** is incorporated; and <u>Abbott et al.</u> further disclose:

- wherein the step of creating is performed by calling at least one function from said first runtime environment *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420). ").*

As per **Claim 55**, the rejection of **Claim 52** is incorporated; and Abbott et al. further disclose:

- wherein the step of creating is performed in the same runtime environment as said application *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420).").*

As per **Claim 56**, the rejection of **Claim 52** is incorporated; and Abbott et al. further disclose:

- wherein the step of creating is performed in a different runtime environment from said application *(see Column 9: 55-58, "Thus the method commences with the start of the Java application whose state is to be saved (step 410), which in turn leads in standard fashion to the initialisation of the Java VM classes (step 420).").*

As per **Claim 57**, the rejection of **Claim 52** is incorporated; and Abbott et al. further disclose:

- executing portions of the application marked for execution prior to said creating step *(see Column 10: 16-21, "... it may be desirable to perform some relatively trivial operation on the classes, such as accessing a class variable, prior to the save step (step 440), in order to ensure that the generic initialisations are indeed completed before the save operation.").*

As per **Claim 58**, Abbott et al. disclose:

- receiving from a runtime environment a serialized representation of objects in a memory space of the runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.")*; and

- building an object code file using the serialized representation and a compiled object code file used to create the memory space *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.").*

As per **Claim 59**, the rejection of **Claim 58** is incorporated; and Abbott et al. further disclose:

- providing the compiled object code file used to create the memory space to the runtime environment prior to said step of receiving *(see Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application.").*

As per **Claim 60**, the rejection of **Claim 58** is incorporated; and Abbott et al. further disclose:

- initializing a serialization process in a separate memory space to create said serialized representation *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

As per **Claim 61**, the rejection of **Claim 58** is incorporated; and Abbott et al. further disclose:

- initializing a serialization process in the runtime environment to create said serialized representation *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).").*

As per **Claim 62**, Abbott et al. disclose:

- creating a serialized representation of application objects in a runtime environment *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 9: 10-20, "... the preferred embodiment of the present invention provides a set of callable routines and tools that allow a user to store away the state of an initialised Java application. The stored application is available for subsequent reloading, but without the start-up cost of initialisation.")*;

- building an object code file using the serialized representation *(see Column 18: 31-35, "The table of object lengths and reference chains is used to reconstruct the object references inside each saved object. Class objects are also loaded at this time and their method tables are restored. This may involve reloading and rebuilding of JIT-compiled code.")*; and

- providing the object code file to a new runtime environment via the network *(see Column 3: 55-60, "... the snapshot can be transmitted over a network, thereby allowing resumption on another system, for example for diagnostic purposes."; Column 18: 61-65, "... the suspended application is now ready to run, and control is passed to execution engine of the Java VM (step 860).")*.

Claim Rejections - 35 USC § 103

14. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

15. **Claims 3, 40, and 63-65** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Abbott et al. (US 7,191,441)** in view of **Vachuska et al. (US 2004/0044989)**.

As per **Claim 3**, the rejection of **Claim 1** is incorporated; however, Abbott et al. do not disclose:

- enumerating a description of each object of the computer application using reflection.

<u>Vachuska et al.</u> disclose:

- enumerating a description of each object of the computer application using reflection *(see Paragraph [0024], "The package 'java.lang.reflect' provides classes and interfaces for obtaining reflective information about classes and objects.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of <u>Vachuska et al.</u> into the teaching of <u>Abbott et al.</u> to include enumerating a description of each object of the computer application using reflection. The modification would be obvious because one of ordinary skill in the art would be motivated to make it possible to examine the structure of the object *(see <u>Vachuska et al.</u> – Paragraph [0004]).*

As per **Claim 40**, the rejection of **Claim 38** is incorporated; however, <u>Abbott et al.</u> do not disclose:

- enumerating a description of each object of an application using reflection.

<u>Vachuska et al.</u> disclose:

- enumerating a description of each object of an application using reflection *(see Paragraph [0024], "The package 'java.lang.reflect' provides classes and interfaces for obtaining reflective information about classes and objects.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of <u>Vachuska et al.</u> into the teaching of <u>Abbott et al.</u> to include enumerating a description of each object of an application using reflection. The modification would be obvious because one of ordinary skill in the art would be motivated to make it possible to examine the structure of the object *(see Vachuska et al. – Paragraph [0004]).*

As per **Claim 63**, the rejection of **Claim 62** is incorporated; however, Abbott et al. do not disclose:

- enumerating a description of each object of the application using reflection.

Vachuska et al. disclose:

- enumerating a description of each object of the application using reflection *(see Paragraph [0024], "The package 'java.lang.reflect' provides classes and interfaces for obtaining reflective information about classes and objects.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Vachuska et al. into the teaching of Abbott et al. to include enumerating a description of each object of the application using reflection. The modification would be obvious because one of ordinary skill in the art would be motivated to make it possible to examine the structure of the object *(see Vachuska et al. – Paragraph [0004]).*

As per **Claim 64**, the rejection of **Claim 63** is incorporated; and Abbott et al. further disclose:

- wherein the new runtime environment is a virtual machine *(see Figure 2: 40).*

As per **Claim 65**, the rejection of **Claim 64** is incorporated; however, Abbott et al. do not disclose:

- wherein the virtual machine is a Flash® renderer.

Official Notice is taken that it is old and well known within the computing art to utilize Flash® as a renderer. The Flash® Player is a client media presentation application available in most common Web browsers. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein the virtual machine is a Flash® renderer. The modification would be obvious because one of ordinary skill in the art would be motivated to execute Flash® files.

16. **Claims 5, 9, 10, 24, 26, 35, 43, 46, and 65** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Abbott et al. (US 7,191,441)**.

As per **Claim 5**, the rejection of **Claim 4** is incorporated; however, Abbott et al. do not disclose:

- wherein the virtual machine is a Flash® renderer.

Official Notice is taken that it is old and well known within the computing art to utilize Flash® as a renderer. The Flash® Player is a client media presentation application available in most common Web browsers. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein the virtual machine is a Flash® renderer. The modification would be obvious because one of ordinary skill in the art would be motivated to execute Flash® files.

As per **Claim 9**, the rejection of **Claim 1** is incorporated; however, Abbott et al. do not disclose:

- wherein the serialized representation is written in an Extensible Markup Language data format.

Official Notice is taken that it is old and well known within the computing art to store data using an Extensible Markup Language (XML) file. XML is a general-purpose markup language that is commonly used to describe data. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein the serialized representation is written in an Extensible Markup Language data format. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

As per **Claim 10**, the rejection of **Claim 9** is incorporated; and Abbott et al. further disclose:

- storing a file prior to said step of building *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization).")*.

However, Abbott et al. do not disclose:

- a markup language file.

Official Notice is taken that it is old and well known within the computing art to store data using an Extensible Markup Language (XML) file. XML is a general-purpose markup language that is commonly used to describe data. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include a markup language file. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

As per **Claim 24**, the rejection of **Claim 22** is incorporated; and Abbott et al. further disclose:

- writing the description to a file and providing the file to the rebuilder *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 18: 10-14, "... the saved state is loaded by a special "javaload" tool, which loads the stored application into a Java VM, bypassing all the initialisation normally associated with bringing up a Java VM. The reloading of the application is performed in stages.")*.

However, Abbott et al. do not disclose:

- an Extensible Markup Language file.

Official Notice is taken that it is old and well known within the computing art to store data using an Extensible Markup Language (XML) file. XML is a general-purpose markup language that is commonly used to describe data. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include an Extensible Markup Language file. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

As per **Claim 26**, the rejection of **Claim 20** is incorporated; however, Abbott et al. do not disclose:

- wherein a virtual machine is a Flash® renderer.

Official Notice is taken that it is old and well known within the computing art to utilize Flash® as a renderer. The Flash® Player is a client media presentation application available in most common Web browsers. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein a virtual machine is a Flash® renderer. The modification would be obvious because one of ordinary skill in the art would be motivated to execute Flash® files.

As per **Claim 35**, the rejection of **Claim 33** is incorporated; however, Abbott et al. do not disclose:

- wherein the runtime memory state is of a Flash® renderer.

Official Notice is taken that it is old and well known within the computing art to utilize Flash® as a renderer. The Flash® Player is a client media presentation application available in most common Web browsers. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein the runtime memory state is of a Flash® renderer. The modification would be obvious because one of ordinary skill in the art would be motivated to execute Flash® files.

As per **Claim 43**, the rejection of **Claim 41** is incorporated; and Abbott et al. further disclose:

- writing a description to a file and compiling a file *(see Column 3: 18-20, "The principle underlying the present invention is serialisation, whereby a "snapshot" is taken of the VM state, which is then stored in a file (serialization)."; Column 4: 21-26, "The preferred embodiment provides the user with various options, as to whether certain actions should be performed prior to determining the current state of the components of the virtual machine. Examples of such actions include performing a garbage collection, and forcing compilation of at least some of the application.").*

However, Abbott et al. do not disclose:

- an Extensible Markup Language file and a markup language file.

Official Notice is taken that it is old and well known within the computing art to store data using an Extensible Markup Language (XML) file. XML is a general-purpose markup language that is commonly used to describe data. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include an Extensible Markup Language file and a markup language file. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

As per **Claim 46**, the rejection of **Claim 39** is incorporated; however, Abbott et al. do not disclose:

- wherein the serialized representation is written in an Extensible Markup Language data format.

Official Notice is taken that it is old and well known within the computing art to store data using an Extensible Markup Language (XML) file. XML is a general-purpose markup language that is commonly used to describe data. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein the serialized representation is written in an Extensible Markup Language data format. The modification would be obvious because one of ordinary skill in the art would be motivated to facilitate the sharing of data across different information systems, particularly via the Internet.

17. **Claim 34** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Abbott et al. (US 7,191,441)** in view of **Wu et al. (US 5,987,256)**.

As per **Claim 34**, the rejection of **Claim 33** is incorporated; however, Abbott et al. do not disclose:

- wherein the object code includes media assets.

Wu et al. disclose:

- wherein the object code includes media assets *(see Column 19: 35-37, "... a standard object file, such as an HTML or JAVA image, is input online 1300 to a compiler 1301 which runs on a standard computer 1302.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Wu et al. into the teaching of Abbott et al. to include wherein the object code includes media assets. The modification would be obvious because one of ordinary skill in the art would be motivated to execute various media contents.

Conclusion

18. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

QC / QC
June 14, 2007

[illegible]
WEI ZHEN
SUPERVISORY PATENT EXAMINER